002/A002, Final Report • June 16, 1992

LOGICS AND MODELS FOR CONCURRENCY AND TYPE THEORY

Prepared by:

José Meseguer, Principal Scientist
Computer Science Laboratory

SRI Project 6729

Prepared for:

Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217-5000

Attn: Dr. Ralph Wachter, Code 1133
Director, Computer Science Division

Contract No. N00014-88-C-0618

Approved:

Mark Moriconi, Director
Computer Science Laboratory

Donald L. Nielson, Vice President
Computing and Engineering Sciences Division

Statement A per telecon
Dr. Ralph Wachter ONR/Code 1133
Arlington, VA 22217-5000

NWW 6/26/92

1 Introduction

Under Office of Naval Research Contract N00014-88-C-0618 (SRI Project 6729), SRI International completed the "Logics and Models for Concurrency and Type Theory" project which was carried out from 1 September 1988 to 30 April 1992.

The goal was to contribute useful new concepts and results in two very active areas of research within semantics of computation, namely concurrency and type theory. The technical method of approach used logic and category theory and aimed at a conceptual unification of concurrency and constructive type theory. Section 2 summarizes the accomplishments attained under this contract and explains the specific ways in which the research goals were met.

Section 3 lists the researchers who participated in the project. The references section lists all the papers written under the contract; copies of the papers are attached to this report.

2 Accomplishments

Several topics were supported by the contract, and there are important connections between these different topics. General logics, often in the particular form of categorical logics, form a common semantic framework for all these investigations; they are discussed in Section 2.1. Work on concurrency models is discussed in Section 2.2. Rewriting logic, its role in unifying models of concurrency, and the Maude language are discussed in Section 2.3. The connections between linear logic and concurrency as well as models for linear logic are discussed in Section 2.4. Sections 2.5, 2.6, and 2.7 cover work on different aspects of type theory, namely models of polymorphism, constructors and selectors, and higher-order subtypes.

2.1 General Logics

The connections between logic and computer science are growing rapidly and are becoming deeper. Besides theorem proving, logic programming, and program specification and verification, other areas showing a fascinating mutual interaction with logic include type theory, concurrency, artificial intelligence, complexity theory, databases, operational semantics, and compiler techniques. The concepts presented in the paper [11] by Meseguer are motivated by the need to understand and relate the many different logics currently being used in computer science, and by the related need for new approaches to the rigorous design of computer systems. Logic programming is of course one of the areas where logic and computer science interact most strongly. The attempt to better understand the nature of this interaction, as well as its future prospects, motivates the following basic question:

What is Logic Programming?

The paper [11] tries to make precise the meaning of this question, and to answer it in terms of general axioms that apply to a wide variety of different logics. In doing so, we are inevitably led to ask the more fundamental question:

What is a Logic?

That is, how should general logics be axiomatized? This is because an axiomatic notion of logic programming must necessarily rest on an axiomatic notion of logic itself. Most of the paper [11] is devoted to the second question. With an axiomatic notion of logic already in place, it then answers the first.

Beyond their application to logic programming, the axioms for general logics given in [11] are sufficiently general to have wide applicability within logic and computer science. Thus, the work reported in [11] has goals that are in full agreement with those of J.A. Goguen and R. Burstall's theory of institutions; however, it addresses proof-theoretic aspects not addressed by institutions. In fact, institutions can be viewed as the model-theoretic component of the theory developed in [11]. The main new contributions include a general axiomatic theory of *entailment* and *proof*, to cover the proof-theoretic aspects of logic and the many proof-theoretic uses of logic in computer science; they also include new notions of *mappings* that interpret one logic (or proof calculus) in another, an axiomatic study of *categorical logics*, and the axioms for logic programming.

In the paper [18], the above-mentioned theory of general logics and its associated axiomatic notion of "logic programming language" are used for defining multiparadigm logical languages. The problem of designing multiparadigm logic programming languages that overcome the present limitations faced by relational and functional logical languages in dealing with state change and reactive systems is approached by a method based on the use of mappings between logics to guide the search for a logic in which the desired multiparadigm integration can be attained. Following this method, rewriting logic is proposed as a logic in which the functional, relational, and concurrent object-oriented paradigms can be unified in a simple and rigorous way. Two languages based on this logic, Maude and MaudeLog, are briefly described and illustrated with examples. Rewriting logic and the Maude language are further discussed in Section 2.3

As already mentioned, *categorical logics* appear as the semantic basis for many of the investigations in this contract. In particular, they are essential for the topics discussed in Sections 2.3, 2.4, 2.5, and 2.7.

2.2 Concurrency Models

Petri nets are widely used to model concurrent systems. However, their composition and abstraction mechanisms are inadequate. The paper [20] by

Meseguer and Montanari solves this problem in a satisfactory way. It views place/transition (P/T) Petri nets as ordinary, directed graphs equipped with two algebraic operations corresponding to parallel and sequential composition of transitions. A distributive law between the two operations captures a basic fact about concurrency. New morphisms are defined, mapping single, atomic transitions into whole computations, thus relating system descriptions at different levels of abstraction. Categories equipped with products and coproducts (corresponding to parallel and nondeterministic compositions) are introduced for Petri nets with and without initial markings. This approach also yields function spaces and new interpretations of duality and invariants. These results provide a formal basis for expressing the semantics of concurrent languages in terms of Petri nets. They also provide a new understanding of concurrency in terms of algebraic structures over graphs and categories that should apply to other models and contribute to the conceptual unification of concurrency.

Descriptions of concurrent behaviors in terms of partial orderings (called *nonsequential processes* or simply *processes* in Petri net theory) have been recognized as superior when information about distribution in space, about causal dependency, or about fairness must be provided. However, at least in the general case of place/transition nets, the proposed models lack a suitable, general notion of *sequential composition.* In the paper [3] by Degano, Meseguer and Montanari, a new algebraic axiomatization is proposed, where, given a net N, a term algebra $\mathcal{P}[N]$ with two operations of parallel and sequential composition is defined. The congruence classes generated by a few simple axioms are proved isomorphic to a slight refinement of classical processes. Actually, $\mathcal{P}[N]$ is a symmetric monoidal category, parallel composition is the monoidal operation on morphisms, and sequential composition is morphism composition. Besides $\mathcal{P}[N]$, we introduce a category $\mathcal{S}[N]$ containing the classical occurrence and step sequences. The term algebras of $\mathcal{P}[N]$ and of $\mathcal{S}[N]$ are in general incomparable, and thus we introduce two more categories $\mathcal{K}[N]$ and $\mathcal{T}[N]$ providing a most concrete and a most abstract extremum, respectively. A simple axiom expressing the functoriality of parallel composition allows us to map $\mathcal{K}[N]$ to $\mathcal{P}[N]$ and $\mathcal{S}[N]$ to $\mathcal{T}[N]$, while commutativity of parallel composition maps $\mathcal{K}[N]$ to $\mathcal{S}[N]$ and $\mathcal{P}[N]$ to $\mathcal{T}[N]$. Morphisms of $\mathcal{K}[N]$ constitute a new notion of concrete net computation, while the strictly symmetric monoidal category $\mathcal{T}[N]$ was introduced previously in [20] as a new algebraic foundation for P/T nets. In the paper [3], the morphisms of $\mathcal{P}[N]$ are proved isomorphic to the processes recently defined in terms of the "swap" transformation by Best and Devillers. Thus, the diamond of the four categories gives a full account in algebraic terms of the relations between interleaving and partial ordering observations of P/T net computations. The paper [4] by Degano, Meseguer, and Montanari provides a full account of previous work by the same authors in [3] on the algebraic axiomatization of concurrent behaviors.

Although place/transition Petri nets are among the most widely used models of concurrency, they still lack a satisfactory semantics: on the one hand the "token game" is too intensional, even in its more abstract interpretations in term of nonsequential processes and monoidal categories; on the other hand, Winskel's basic unfolding construction, which provides a coreflection between nets and finitary prime algebraic domains, works only for safe nets. The paper [21] by Meseguer, Montanari, and Sassone improves this situation by extending Winskel's result to P/T nets. The work begins defining a rather general category **PTNets** of P/T nets; then, a category **DecOcc** of decorated (nondeterministic) occurrence nets is introduced and adjunctions between **PTNets** and **DecOcc** and between **DecOcc** and **Occ**, the category of occurrence nets, are defined. The role of **DecOcc** is to provide natural unfoldings for P/T nets, that is, acyclic safe nets where a notion of family is used for relating multiple instances of the same place. The unfolding functor from **PTNets** to **Occ** reduces to Winskel's when restricted to safe nets, while the standard coreflection between **Occ** and **Dom**, the category of finitary prime algebraic domains, when composed with the unfolding functor above, determines a chain of adjunctions between **PTNets** and **Dom**.

An additional development also related to the partial order or "true concurrency" approach to concurrency is the far-reaching generalization of partially ordered computations (which are based on the very simple temporal structure of precedence of one event by another) to computations endowed with much richer temporal structures such as real time, interval time, or probabilistic time that is given in the paper [1] by Casley, Crew, Meseguer, and Pratt. The framework is indeed very general; it uses a deep category-theoretic insight of F.W. Lawvere realizing that enriched categories over a symmetric monoidal category are generalized metric spaces. It is precisely by using this insight that widely different temporal structures can be studied within a common framework, and that basic constructions for concurrent computations can in fact be made independent of the particular temporal structure chosen. In this way, the relevant notion of time can be made into a parameter of the basic constructions, and the different levels of description (corresponding to different notions of time) can be systematically related. If only an order relation between events is relevant, we have the special case of *pomset* computations, but if, for example, timing is important, duration constraints given by real numbers can be introduced in an abstract description of the computation. The relevant mathematical structure is that of a $\mathcal{D}$-category that essentially[1] formalizes, for an appropriate choice of time domain $\mathcal{D}$—where the time domain $\mathcal{D}$ is formalized as a monoidal category—the desired general notion of concurrent computation.

[1]The situation is actually somewhat more complicated, due to a labeling of the events contained in the computation that is typically added and that requires some additional structure.

The paper [2] by Casley, Crew, Meseguer, and Pratt is the final version of the above-mentioned work by the same authors.

2.3 Rewriting Logic and the Unification of Concurrency Models

The main goal of the paper [15] by Meseguer is to propose a general and precise answer to the question:

What is a concurrent system?

It seems fair to say that this question has not yet received a satisfactory answer, and that the resulting situation is one of *conceptual fragmentation* within the field of concurrency. A related problem is the *integration of concurrent programming with other programming paradigms*, such as functional and object-oriented programming. Integration attempts typically graft an existing concurrency model on top of an existing language, but such *ad hoc* combinations often lead to monstrous deformities that are extremely difficult to understand. Instead, the paper proposes a *semantic integration* of those paradigms based on a common *logic and model theory*.

The logic, called *rewriting logic*, is implicit in term rewriting systems but has passed for the most part unnoticed, due to our overwhelming tendency to associate term rewriting with equational logic. Its proof theory exactly corresponds to (truly) concurrent computation, and the model theory proposed for it in this paper provides the general concept of concurrent system that we are seeking.

The paper also proposes rewrite rules as a very high-level language to program concurrent systems. Specifically, a language design based on rewriting logic is presented containing a *functional sublanguage* entirely similar to OBJ3 as well as more general *system modules*, and also *object-oriented modules* that provide notational convenience for object-oriented applications but are reducible to system modules [14]. The language's semantics is directly based on the model theory of rewriting logic and yields the desired semantic integration of concurrency with functional and object-oriented programming.

The resulting notion of concurrent system is indeed very general and specializes to a wide variety of existing notions in a very natural way, including labeled transition systems, Petri nets, concurrent object-oriented programming, and several others. Such specializations, as well as the extension of the ideas to the case of conditional rewrite rules, are studied and discussed in much greater detail in the technical report [16].

The papers [17, 13] develop rewriting logic as a concurrent model of computation supporting a very general style of declarative programming. Rewriting with conditional rewrite rules modulo a set E of structural axioms provides a general framework for unifying a wide variety of models of

concurrency including Petri nets, CCS, Actors, concurrent object-oriented programming, the UNITY model of computation, and parallel functional programming. Concurrent rewriting coincides with logical deduction in *conditional rewriting logic*, a logic of actions whose models are concurrent systems. This logic is sound and complete and has initial models. In addition to general models interpreted as concurrent systems that provide a more operational style of semantics, more restricted semantics with an increasingly denotational flavor such as preorder, poset, cpo, and standard algebraic models appear as special cases of the model theory. This permits dealing with operational and denotational issues within the same model theory and logic. A programming language called Maude whose modules are rewriting logic theories is defined and given denotational and operational semantics. Maude provides a simple unification of concurrent programming with functional and object-oriented programming and supports high-level declarative programming of concurrent systems.

Object-oriented Concurrency

Despite the growing interest in object-oriented programming in general and object-based concurrency in particular, many unresolved research issues remain and it seems important to seek a simple and general semantic basis on which rigorous progress in this subject can be based. The papers [14, 17] contain a specific proposal for a semantic basis that could serve these purposes. They use rewriting logic to provide a simple and general semantics for object-oriented concurrent systems. Object-based concurrent computation corresponds in this model to logical deduction performed by *concurrent rewriting* modulo structural axioms of associativity, commutativity, and identity that capture abstractly the essential aspects of communication in a distributed object-oriented configuration made up of concurrent objects and messages. Thanks to this axiomatization, it becomes possible to study the behavior of concurrent objects by formal methods in a logic intrinsic to their computation. The relationship with Actors and with other models of concurrent computation is also discussed. The Maude language embodies these ideas and serves as a vehicle to illustrate the basic concepts by means of examples. Maude has three types of modules: functional modules (OBJ3 can be viewed as Maude's functional sublanguage, and therefore these are essentially OBJ3 programs); system modules, which denote general concurrent systems; and object-oriented modules, which denote concurrent object-oriented systems. From the mathematical point of view, object-oriented modules are reducible to system modules, but they have a special syntax to support object-oriented design.

Parallel Programming in Maude

The paper [22] by Meseguer and Winkler explores the parallel programming and wide spectrum aspects of Maude, which, as already mentioned, is a declarative parallel programming language based on rewriting logic. Parallelism in Maude is implicit; it is based on the intrinsically parallel nature of logical deduction in rewriting logic. Maude unifies functional programming, concurrent object-oriented programming, and general concurrent systems programming within a single logic. *Functional modules* form a sublanguage essentially identical to the OBJ language, and *object-oriented modules* provide convenient syntax for object-oriented applications, but are translatable into more general *system modules*. Maude is a *wide-spectrum language* that integrates nonexecutable specifications, executable specifications for rapid prototyping, and machine-independent, efficiently implementable parallel code written in a sublanguage called Simple Maude. Simple Maude's *machine independence*—due to the flexibility and generality of its logical model of concurrent computation—makes it a good candidate for implementations in MIMD, SIMD, and MIMD/SIMD architectures. Simple Maude also supports *multilingual extensions*, allowing reuse and parallelization of conventional code that can be incorporated in "black box" modules.

2.4 Linear Logic and Concurrency

Linear logic has been recently introduced by Girard as a logic of actions that seems well suited for concurrent computation. In the papers [5, 9] by Martí-Oliet and Meseguer, a systematic correspondence between Petri nets, linear logic theories, and linear categories is established. Such a correspondence sheds new light on the relationships between linear logic and concurrency, and on how both areas are related to category theory. Categories are here viewed as concurrent systems whose objects are states, and whose morphisms are transitions. This is an instance of the Lambek-Lawvere correspondence between logic and category theory that cannot be expressed within the more restricted framework of the Curry-Howard correspondence.

Martí-Oliet and Meseguer gave a new algebraic axiomatization of linear logic models in [8], leading to substantial simplifications in the final version of [5]. The new axioms directly reflect at the model-theoretic level the de Morgan duality exhibited by linear logic, and are considerably simpler than previous axioms. Several equationally defined classes of models have been studied. One such class suggests a new variant of linear logic, called cancellative linear logic, in which it is always possible to cancel a proposition (viewed as a resource) and its negation (viewed as a debt.) This provides a semantics for a generalization of the usual token game on Petri nets, called financial game. Poset models, called Girard algebras, are also defined equationally; they generalize for linear logic the Boolean algebras of classical logic, and contain the quantale models as a special case. The proposed ax-

iomatization also provides a simple set of categorical combinators for linear logic, extending those previously proposed by Lafont.

The categorical foundations of this new axiomatization of linear logic models were the subject of a separate study by Martí-Oliet and Meseguer in [6]. A key concept is that of a dualizing object in a closed monoidal category. This notion is important for the categorical semantics of linear logic, where dualization corresponds to negation, and for the fields of linear algebra and topological vector spaces, where dualities of this form are systematically exploited. The paper [6] develops an axiomatic theory of duality based on the notion of a dualizing object, discusses a variety of examples, and studies the important case in which, in addition, there is a natural isomorphism between the functor corresponding to the connective "par" and the tensor product functor. The paper also contains a detailed comparison between the notion of a category with a dualizing object and Barr's notion of $*$-autonomous category, and concludes that dualizing objects provide a better axiomatic basis for the treatment of duality.

The paper [10] surveys recent work on the applications of linear logic to concurrency, with special emphasis on Petri nets and on the use of categorical models. In particular, it presents a synthesis of previous work by Martí-Oliet and Meseguer on the systematic correspondence between Petri nets, linear logic theories, and linear categories, and explain its relationships to work by many other authors. Throughout, the computational interpretation of the linear logic connectives is discussed and the ideas are illustrated with examples.

Categories play an important role in this survey. On the one hand, from a computational perspective, they are interpreted as concurrent systems whose objects are states, and whose morphisms are transitions; on the other hand, when a model-theoretic perspective is adopted, they provide a very flexible conceptual framework within which the relationships among quite different models already proposed for linear logic can be better understood; this framework also suggests the study of new models and an axiomatic treatment of classes of models. The categorical semantics for linear logic is based on dualizing objects and permits a very simple presentation of ideas requiring a more complicated treatment in the language of $*$-autonomous categories.

The survey is based on the previous paper [9], which has been greatly extended in several ways. First, a detailed comparison between the concepts of category with a dualizing object and Barr's $*$-autonomous category has been added.

Second, the basic categorical context in which the semantics of linear logic should be discussed is that of a closed symmetric monoidal category[2].

[2]For noncommutative linear logic, the broader context of closed nonsymmetric monoidal categories in the style of Lambek should be adopted; this paper concentrates on the symmetric case. It is also possible to give the notion of a dualizing object in the

Developing the work reported in this paper has presented the difficulty of not having an easy source of reference, suitable for computer scientists, for basic concepts and properties about closed (symmetric monoidal) categories, although the basic reference still remains the original monograph of Eilenberg and Kelly. Therefore, the paper includes a fully self-contained exposition of closed symmetric monoidal categories in an appendix. The survey also contains results on equationally defined classes of models for linear logic that previously appeared in [8].

Third, comparisons with the work done in this area by several researchers in the time elapsed since the first version of [9] was written are included. As already mentioned, the survey focuses on the relationship between linear logic and concurrency theory with special emphasis on Petri net theory, without trying to cover other areas. However, the concluding remarks discuss various other areas of application and suggest some relevant references for those other areas to the interested reader.

2.5 Relating Models of Polymorphism

To meet the software crisis, programming language design strives for principles and concepts that support increasingly higher levels of code reuse. Of particular importance are techniques that allow the development of complex modules by combining preexisting ones in a systematic way. We can conceive of such combinations as providing an *algebra* of modules, that mirrors at a very high level the low-level algebraic character of, say, arithmetic expressions. Modules themselves are the values, and the analogue role of operators such as addition or multiplication is played by *generic* modules that take one or more modules as arguments and yield a complex module as a result. This can be accomplished in a variety of ways, based on different logics. For example, in the context of the first-order functional language OBJ, generic modules are understood as algebraic theories having specified parameter subtheories, and a very rich algebra of "module expressions" is obtained by "putting theories together" as in the language Clear. In this way, first-order generic modules provide higher-order programming capabilities. This paper is concerned with the alternative, *explicitly higher-order* approach pioneered by John Reynolds, whose logical aspects were independently investigated for other purposes by the logician J.-Y. Girard. This approach is known as the second-order polymorphic lambda calculus (abbreviated λ_2). In it, generic modules appear as *polymorphic functions* that take types as arguments. This calculus plays a central role in higher-order functional programming, and many other type theories can be viewed as extensions of it.

---

general case of closed categories, without a tensor product; from a proof-theoretic point of view, this could be useful for the study of fragments of linear logic that include the $\multimap$ connective but not the $\otimes$ connective.

Many different notions of model have been proposed for λ_2, and it seems fair to say that there is as yet no final agreement on the matter, and that the relationships between the different models have not been sufficiently clarified. This is an unsatisfactory situation for a topic of great importance. The paper [12] by Meseguer presents some new ideas and results that help in gaining a more unified view of the semantics of polymorphism and in better understanding the relationships between different approaches. This is accomplished by establishing semantic relationships at three different levels:

1. At the level of **models**, by relating models by means of homomorphisms. In particular, a new *initial model semantics* for polymorphism is given for the basic calculus and for several different extensions to richer calculi.

2. At the level of different **notions** of model, by relating their semantics. Technically, this takes the form of *functors* among different categories of models.

3. At the level of **type theories**, by relating the second-order polymorphic lambda calculus to its natural extensions, when fixpoints or Type:Type are added, and also to Martin-Löf type theory (abbreviated $\lambda\Pi$). This takes the form of a *map* between logics that either brings each λ_2 theory into an appropriate extension of λ_2, or translates it into a corresponding theory in $\lambda\Pi$.

Besides establishing such relationships, the work reported in [12] tries to recover the original intuition of a model of λ_2 as a *universe*, an intuition that Reynolds has shown cannot be maintained within classical set theory, and that is lost or obscured in more esoteric notions of model. However, by adopting the constructive notion of set advocated by Per Martin-Löf, all foundational contradictions disappear and polymorphism is intuitionistically set theoretic. In this way, the naive notion of a *universe model* can be maintained, and a general categorical semantics can be developed. Also, the notion extends very nicely to richer calculi that add fixpoints or a type of all types to λ_2. Even though some of those richer calculi are *not* set theoretic (not even intuitionistically), they can be given a categorical, initial model, semantics in a context generalizing that of the basic calculus.

2.6 Subtypes, Constructors and Selectors

Structured data are generally composed from constituent parts by constructors and decomposed by selectors. In the paper [19] (an extensively revised and improved new version of an earlier conference paper) Meseguer and Goguen show that the usual many-sorted algebra approach to abstract data types cannot capture this simple intuition in a satisfactory way. They also show that order-sorted algebra does solve this problem, and many others

concerning partially defined, ill-defined and erroneous expressions, in a simple and natural way. In particular, it is shown how order-sorted algebra supports an elegant solution to the problems of multiple representations and coercions. The essence of order-sorted algebra is that sorts have subsorts, whose semantic interpretation is the subset relation on the carriers of algebras.

2.7 Higher-order Subtypes

The failure to make explicit two different notions of subtype, a *subtype as inclusion* notion originally proposed by Goguen and a *subtype as implicit conversion* notion originally proposed by Reynolds, leads to unsatisfactory situations in present approaches to subtyping. In fact, these two lines of work have had very little mutual interaction, and—with a few exceptions—almost nothing has been done to compare their relative strengths and weaknesses. We are convinced that much can be gained, by way of mutual enrichment, from such a comparison, and the paper [7] by Matí-Oliet and Meseguer should be seen as a step in this direction. We argue that choosing either notion at the expense of the other would be mistaken and limiting, and propose a framework in which two subtype relations $\tau \leq \tau'$ (inclusion) and $\tau \leq: \tau'$ (implicit conversion) are distinguished and integrated.

For example, one of the nicest features of the subtype as inclusion notion is that it is *completely safe* to move data and perform operations up and down the subtype hierarchy, so that for all purposes we can ignore what type we are at. This subtype notion is probably the most natural and the most widely held, and agrees perfectly well with traditional practice and notation in mathematics, where we can for example add the number 3 to the complex expression $(-i) * i$ and then evaluate the whole expression to the natural number 4, or we can instead first evaluate $(-i) * i$ to 1 and then add the natural numbers 3 and 1 to get 4 as a result. This safety in moving data up and down is guaranteed by the following "no loss of information" axiom: if $\tau \leq \tau'$,

$$\forall x, y : \tau \qquad x =_\tau y \iff x =_{\tau'} y$$

which is typically implicit in treatments such as order-sorted algebra, where the equality relation is defined independently of particular typings.

By contrast, such safety is not possible in the implicit conversion approach, for which the above axiom fails even in the case where the subtype relations on basic types are all inclusions. This can be illustrated by the rule for function spaces

$$(\Rightarrow) \quad \frac{\tau \leq \tau' \quad \rho \leq \rho'}{(\tau' \Rightarrow \rho) \leq (\tau \Rightarrow \rho')}$$

originally due to Reynolds, which is typical of higher-order approaches to subtyping.

The main point to emphasize is that *two quite different semantic intuitions are being conflated under the term "subtype,"* namely, the inclusion and the implicit conversion notions. We think that it would be a serious mistake to think that one has to *choose* one of these two notions at the expense of the other; actually, either choice would have undesirable consequences. For example, the nice preservation of information properties of the inclusion notion and the associated intuitions and ease in manipulating data would be lost if we side with implicit conversions; but insisting on inclusions as the only relevant notion would also be undesirable, since we would lose the nice ability supported by the rule ($\Rightarrow$) of passing as arguments functions having a bigger domain of definition than strictly required.

Most of the paper [7] is devoted to extending the first-order theory of subtypes as inclusions already developed in work on order-sorted algebra by Goguen and Meseguer to a higher-order context; this involves providing a higher-order equational logic for (inclusive) subtypes, a categorical semantics for such a logic that is complete and has initial models, and a proof that this higher-order logic is a conservative extension of its first-order counterpart. We then give axioms that integrate the $\leq$ and $\leq:$ relations in a unified categorical semantics. Besides enjoying the benefits provided by each of the notions without their respective limitations, our framework supports rules for structural subtyping that are more informative and can discriminate between inclusions and implicit conversions.

3 Personnel

The project was led by Dr. José Meseguer. The following researchers have also worked on the project; for those who were visitors, their permanent affiliation is given. Dr. Martí-Oliet finished his doctorate under the supervision of Dr. Meseguer, thanks to the funding of this project. Mr. Sassone and Ms. Cerioli are graduate students whose doctoral thesis work will include research also funded under this project.

- Mr. Timothy Winkler
- Dr. Narciso Martí-Oliet
- Prof. Ugo Montanari, University of Pisa, Italy
- Prof. Pierpaolo Degano, University of Parma, Italy
- Mr. Vladimiro Sassone, University of Pisa, Italy
- Ms. Maura Cerioli, University of Genova, Italy

References

[1] Ross Casley, Roger Crew, José Meseguer, and Vaughan Pratt. Temporal structures. In D.H. Pitt et al., editor, *Category Theory and Computer Science*, pages 21–51. Springer LNCS, Vol. 389, 1989. Extended version to appear in *Mathematical Structures in Computer Science.*

[2] Ross Cassley, Roger Crew, José Meseguer, and Vaughan Pratt. Temporal structures. *J. Math. Structures in Computer Science*, 1(2):179–213, 1991.

[3] P. Degano, J. Meseguer, and U. Montanari. Axiomatizing net computations and processes. In *Proc. LICS'89*, pages 175–185. IEEE, 1989.

[4] P. Degano, J. Meseguer, and U. Montanari. Axiomatizing the algebra of net computations and processes. Technical Report SRI-CSL-90-12, SRI International, Computer Science Laboratory, November 1990. Submitted for publication.

[5] Narciso Martí-Oliet and José Meseguer. From Petri nets to linear logic. In D.H. Pitt et al., editor, *Category Theory and Computer Science*, pages 313–340. Springer LNCS 389, 1989. Final version in *Mathematical Structures in Computer Science*, 1:69-101, 1991.

[6] Narciso Martí-Oliet and José Meseguer. Duality in closed and linear categories. Technical Report SRI-CSL-90-01, SRI International, Computer Science Laboratory, February 1990.

[7] Narciso Martí-Oliet and José Meseguer. Inclusions and subtypes. Technical Report SRI-CSL-90-16, SRI International, Computer Science Laboratory, December 1990. Submitted for publication.

[8] Narciso Martí-Oliet and José Meseguer. An algebraic axiomatization of linear logic models. In G.M. Reed, A.W. Roscoe, and R. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 335–355. Oxford University Press, 1991. Also Technical Report SRI-CSL-89-11, SRI International, Computer Science Laboratory, December 1989.

[9] Narciso Martí-Oliet and José Meseguer. From Petri nets to linear logic. *Math. Struct. in Comp. Sci.*, 1:69–101, 1991.

[10] Narciso Martí-Oliet and José Meseguer. From Petri nets to linear logic through categories: a survey. *Intl. J. of Foundations of Comp. Sci.*, 2(4):297–399, 1991.

[11] J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

[12] J. Meseguer. Relating Models of Polymorphism. In *Proc. POPL'89*, pages 228–241. ACM, 1989.

[13] José Meseguer. Conditional rewriting logic: deduction, models and concurrency. In S. Kaplan and M. Okada (eds.) *Proc. CTRS'90*, Montreal, Canada, 1990, Springer LNCS 516, pp. 64-91, 1991.

[14] José Meseguer. A logical theory of concurrent objects. In *ECOOP-OOPSLA'90 Conference on Object-Oriented Programming, Ottawa, Canada, October 1990*, pages 101–115. ACM, 1990.

[15] José Meseguer. Rewriting as a unified model of concurrency. In *Proceedings of the Concur'90 Conference, Amsterdam, August 1990*, pages 384–400. Springer LNCS 458, 1990.

[16] José Meseguer. Rewriting as a unified model of concurrency. Technical Report SRI-CSL-90-02, SRI International, Computer Science Laboratory, February 1990. Revised June 1990.

[17] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992. Also, SRI International, Computer Science Laboratory technical report SRI-CSL-91-05, February, 1991.

[18] José Meseguer. Multiparadigm logic programming. To appear in Proc. 3rd Intl. Conf. on Algebraic and Logic Programming, Springer LNCS, 1992.

[19] José Meseguer and Joseph Goguen. Order-sorted algebra solves the constructor-selector, multiple representation and coercion problems. Technical Report SRI-CSL-90-06, SRI International, Computer Science Laboratory, June 1990. To appear in *Information and Computation.*

[20] José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88:105–155, 1990. Appeared as SRI Tech Report SRI-CSL-88-3, January 1988.

[21] José Meseguer, Ugo Montanari, and Vladimiro Sassone. On the semantics of Petri nets. To appear in Proc. Concur'92, Springer LNCS, 1992.

[22] José Meseguer and Timothy Winkler. Parallel Programming in Maude. In J.-P. Banâtre and D. Le Mètayer, editors, *Research Directions in High-level Parallel Programming Languages*, pages 253–293. Springer-Verlag, 1992. LNCS, Volume 574; also, SRI Technical Report SRI-CSL-91-08, November 1991.